XAPP999 (v1.0) February 8, 2008

Reference System: PLBv46 PCI Using the ML555 Embedded Development Platform

Author: Lester Sanders

Summary

This application note describes how to build a reference system for the Processor Local Bus Peripheral Component Interconnect (PLBv46 PCI) core using a MicroBlaze™ processor-based embedded system in the ML555 Embedded Development Platform.

A set of files containing Xilinx Microprocessor Debugger (XMD) commands is provided for writing to the Configuration Space Headers and for verifying that the PLBv46 PCI core is operating correctly. Two software projects illustrate how to configure the PLBv46 PCI cores, scan configuration registers, and set up and use DMA operations. The procedure for using the ChipScope™ Analyzer to analyze PLBv46 PCI and system functionality is provided.

Included Systems

This application note includes one reference system:

www.xilinx.com/support/documentation/application_notes/xapp999.zip

The project name used in xapp999.zip is ml555_mb_plbv46_pci.

Required Hardware and Tools

Users must have the following tools, cables, peripherals, and licenses available and installed. EDK provides an evaluation license for the PLBv46 PCI.

- Xilinx EDK 9.2.02i
- Xilinx ISE™ 9.2.04i
- Xilinx Download Cable (Platform Cable USB or Parallel Cable IV)
- Modeltech ModelSim v6.1e
- Chipscope Analyzer 9.2.01
- PLBv46 PCI License
- USB Type A to Type B Interface Cable
- Communication Utility (HyperTerminal, Tera Term)
- ML555 PCI/PCI Express Evaluation Platform

Introduction

PCI transactions are done between an initiator and a target. This reference design is for the ML555 Evaluation Platform. To be useful, the ML555 Evaluation Platform must be inserted into a PCI slot. In the examples provided in this application note, the ML555 Embedded Development Platform is inserted into PCI slot P3 of the Xilinx ML410 Evaluation Platform. This allows both configuration and memory transactions to be done on the PCI bus between an initiator and a target. The examples use the ML410 PLBv46 PCI as the initiator and the ML555 PLBv46 PCI as the target. It is relatively easy to modify the examples so that the initiator and target functions are swapped.

© 2008 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

Figure 1 is a block diagram of the reference system.

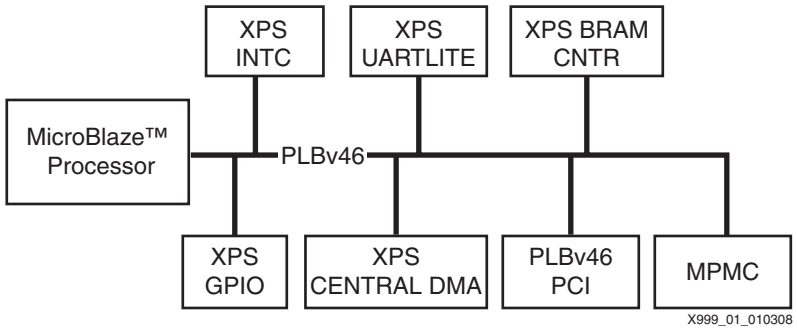


Figure 1: **ML555 PLBv46 PCI Reference System Block Diagram**

Figure 2 shows the principal interface blocks when transferring data between the PLBv46 PCI Bridge in the XC5VLX50T on the ML555 board and the PLBv46 PCI Bridge in the XC4VFX60 on the ML410 board.

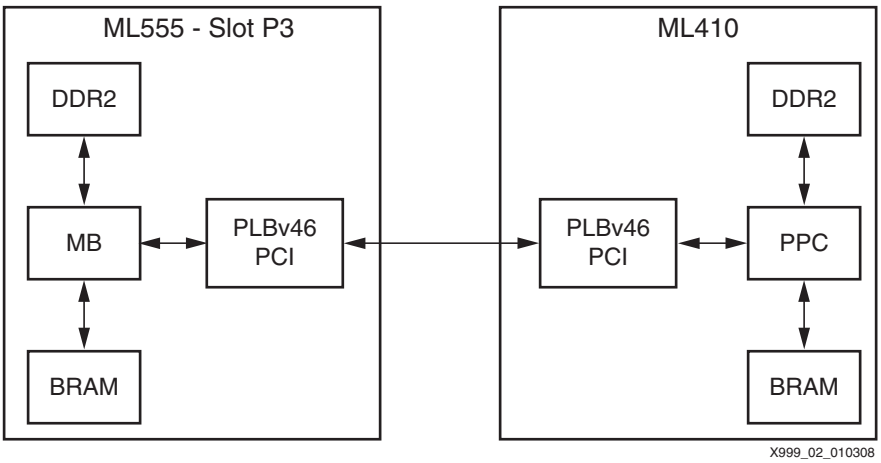


Figure 2: **Interfacing ML555 PLBv46 PCI with ML410 PLBv46 PCI**

ML555 Setup

Figure 3 shows the ML555 Evaluation Platform. The ML555 has a PCI connector on one edge of the PCB and a x8 PCIe connector on the other edge. For PCI operation, move switch SW8 to the PCI position and remove the shunts on P18.

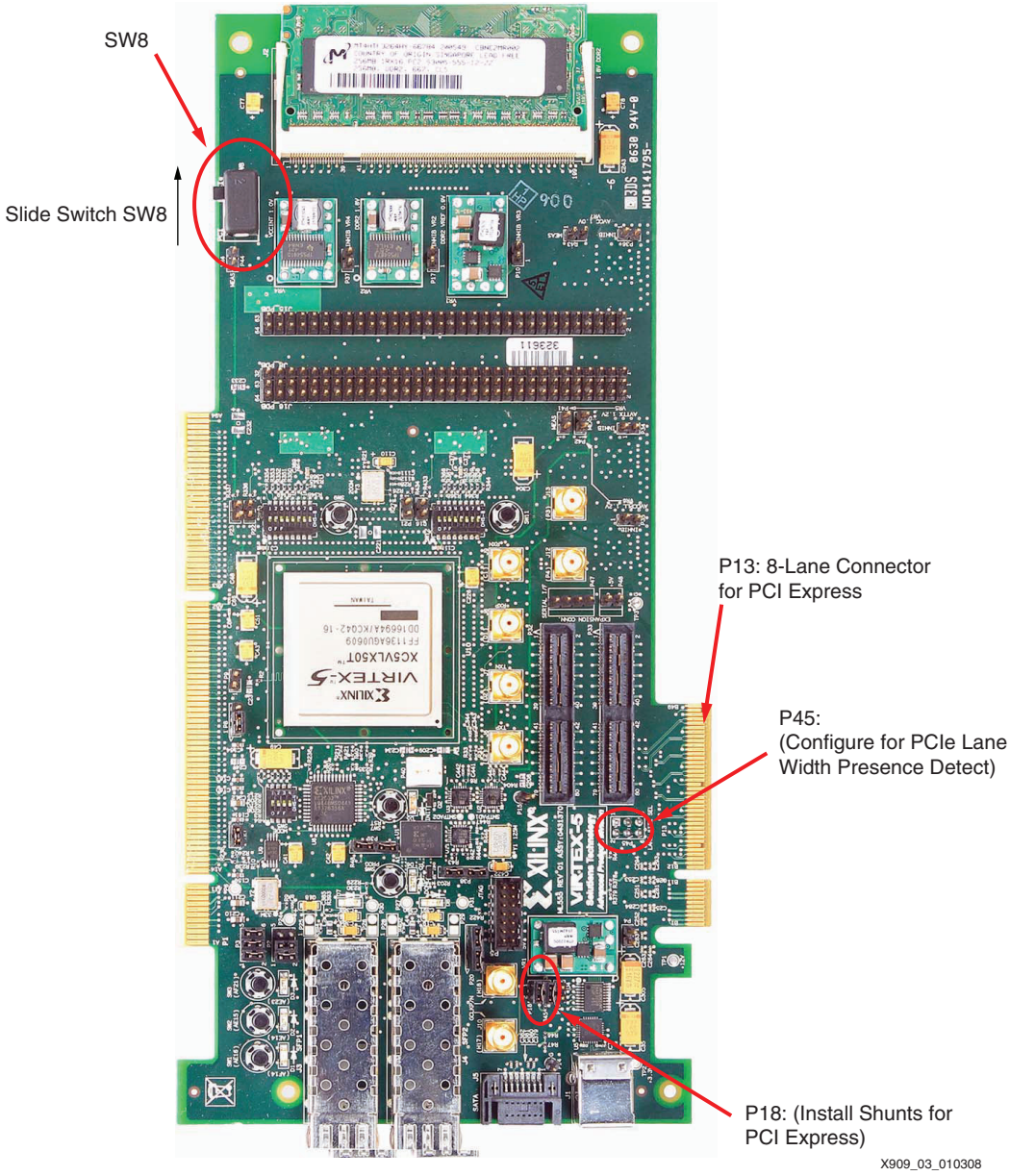

SW8
Slide Switch SW8
P13: 8-Lane Connector for PCI Express
P45: (Configure for PCIe Lane Width Presence Detect)
P18: (Install Shunts for PCI Express)
X909_03_010308

Figure 3: **ML555 PCI/PCI Express Evaluation Platform**

The ML555 includes a 64-bit PCI edge connector, 128 MB (or 256 or 512 MB) DDR2 SDRAM memory, LED displays, XCF32P-FSG48C Platform Flash configuration PROM, and a JTAG port. The MicroBlaze processor runs at 75 MHz.

The PCI functions, devices, and buses in the PCI reference designs are addressed using the Configuration Address Port format shown in Figure 4.

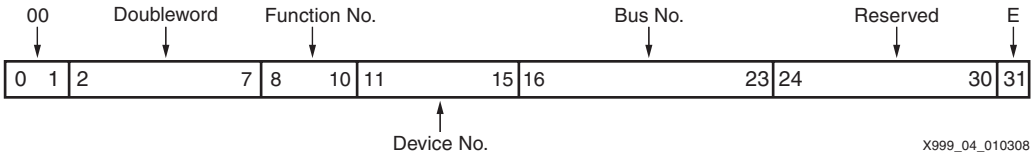


Figure 4: **Configuration Address Port Format**

Implementation Results

Table 1 shows the resource utilization in the reference design.

Table 1: **Design Resource Utilization**

Resources	Used	Available	Utilization (%)
Slice registers	8475	28800	29
Slice LUTs	8843	28800	30
DCM	2	12	16
PLL_ADV	0	6	0
Block RAM	37	60	61

The Configuration Address Port and Configuration Data Port registers in the host bridge PLBv46 PCI Bridge are used to configure multiple PCI bridges when host bridge configuration is enabled.The bit definitions of the Configuration Address Port in the big endian format used by the PLBv46 are given in Table 2.

Table 2: **Configuration Address Port Register Definitions**

Bit	Definition
0-5	Target word address in configuration space
6-7	Hardwired to 0
8-12	Device
13-15	Function
16-23	Bus Number
24	Enable
25-31	Hardwired to 0

Reference System Specifics

In addition to the MicroBlaze processor and PLBv46 PCI, this system includes the cores shown in Figure 1: DDR2, BRAM memory and UART, XPS Central DMA, MDM, GPIO, and an interrupt controller. The PCI Arbiter core is included in the FPGA.

Virtex-5 ML555 PCI/PCI Express Evaluation Platform

In the reference design, the PLBv46 PCI in the XC5VLX50T on the ML555 board interfaces to the PLBv46 PCI in the Virtex-4 ML410 Evaluation Platform. The ML555 board uses the Xilinx XC5VLX50T device in the 1136 pin package.

Table 3 provides the address map for the XC5VLX50T.

Table 3: **ML555 Address Map**

Peripheral	Instance	Base Address	High Address
LMB_BRAM_IF_CNTLR	DLMB_CNTLR/ILMB_CNTLR	0x00000000	0x00001FFF
XPS UartLite	RS232_Uart_1	0x84000000	0x8400FFFF
PLBv46 PCI	plbv46_pci_0	0x42600000	0x426001FF
MPMC	DDR_SDRAM_64Mx32	0x90000000	0x9FFFFFFF
XPS GPIO	LEDs_8Bit	0x81400000	0x8140FFFF
MDM	debug_module	0x84400000	0x8440FFFF
XPS INTC	xps_intc_0	0x80200000	0x8020FFFF
xps_central_dma	xps_central_dma_0	0x81810000	0x8181FFFF
xps_bram_if_cntlr	xps_bram_if_cntlr_1	0x8AE08000	0x8AE0FFFF

The application note XAPP1001 Reference System: PLBv46 PCI in a ML410 Embedded Development Platform provides a link to the hardware design files used for the ML410.

Configuration of PLBv46 PCI on the ML410 Board

When used in a Virtex-4 FPGA, the PLBv46 PCI bridge uses the 32-bit Xilinx LogiCore™ Version 3 (v3.0) core. For the PLBv46 PCI bridge to perform transactions on the PCI bus, the v3.0 core must be configured using configuration transactions from either the PCI-side or the PLBv46 side. In this application note, the ML410 PLBv46 PCI is the host bridge which configures itself and the ML555 PLBv46 PCI. The C_INCLUDE_PCI_CONFIG parameter is set to 1. In this case, IDSEL input of the v3.0 is connected to the address ports specified in Table 2, while the IDSEL port of the bridge is unused.

To write to the configuration header, use the steps listed below.

1. Configure the Command and Status Register. The minimum that must be set is the Bus Master Enable bit in the command register. For memory transactions, set the memory space bit. For I/O transactions, set the I/O space bit.

2. Configure the Latency Timer to a non-zero value.

3. Configure at least one BAR. Configure additional BARs as needed for other memory/IO address ranges.

The v3.0 core configures itself only after the Bus Master Enable bit is set and the latency timer is set to avoid time-outs. If the v3.0 core latency timer remains at the default 0 value, configuration writes to remote PCI devices do not complete, and configuration reads of remote PCI devices terminate due to the latency timer expiration. Configuration reads of remote PCI devices with the latency timer set to 0 return 0xFFFFFFFF.

ML410 XC4VFX60 Address Map

Table 4: **ML410 XC4VFX60 System Address Map**

Peripheral	Instance	Base Address	High Address
MPMC	DDR_SDRAM_32Mx64	0x00000000	0x03FFFFFF
XPS UART16550	RS232_Uart_1	0x83E00000	0x83E0FFFF
XPS INTC	XPS_intc_0	0x81800000	0x8180FFFF
PLBv46_PCI	PCI32_Bridge	0x85E00000	0x85E0FFFF
XPS Central DMA	xps_central_dma_0	0x80200000	0x8020FFFF
XPS BRAM	xps_bram_if_cntlr_0	0xFFFF0000	0xFFFFFFFF
XPS GPIO	LEDs_8Bit	0x81400000	0x8140FFFF
XPS Sysace	SysACE_CompactFlash	0x83600000	0x8360FFFF
XPS IIC	IIC_Bus	0x81600000	0x8160FFFF

Configuration of the PLBv46 PCI on the ML555 Board

The reference design contains the following settings for ML555 PLBv46 PCI generics:

C_FAMILY = virtex5

C_INCLUDE_PCI_CONFIG = 0

C_INCLUDE_BAROFFSET = 0

C_IPIFBAR_NUM = 2

C_PCIBAR_NUM = 2

C_IPIFBAR_0 = 0x20000000

C_IPIFBAR2PCIBAR_0 = 0x80000000

C_IPIFBAR_1 = 0xE8000000

C_IPIFBAR2PCIBAR_1 = 0x90000000

When C_FAMILY is defined as virtex5, the PLBv46 PCI uses the v4.0 PCI LogiCore.

Figure 5 shows how to specify the Base Address Registers (BARs) in XPS.

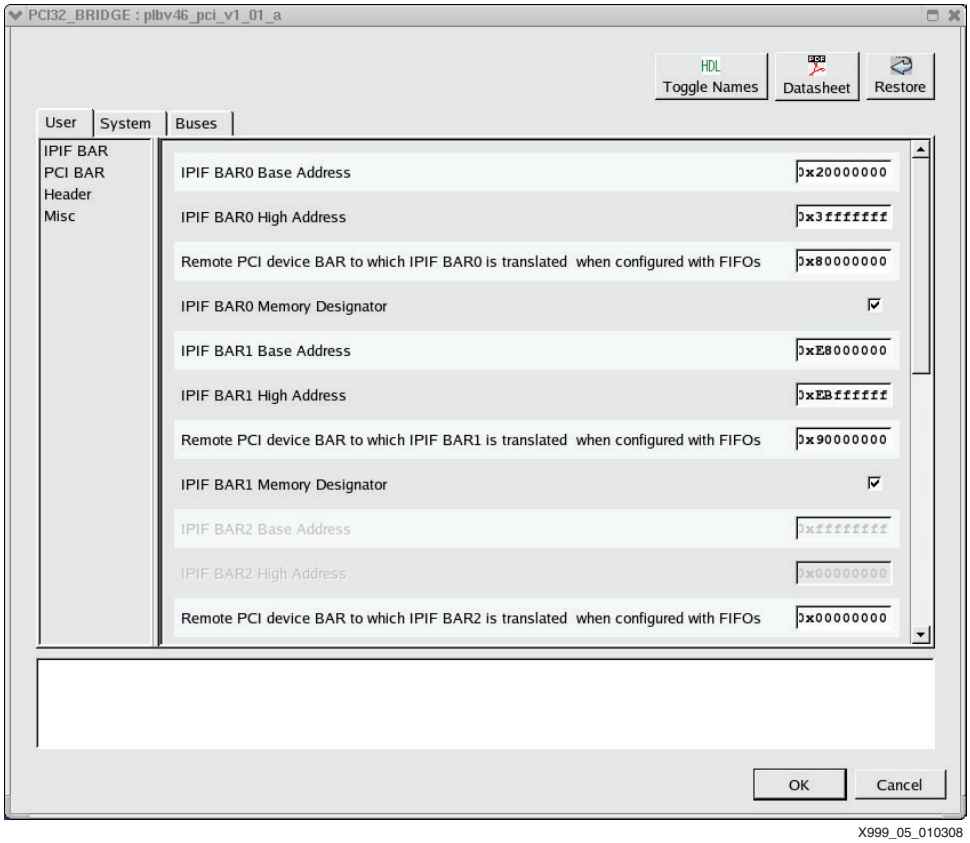


Figure 5: **Specifying the Values of Generics in XPS**

Setting C_INCLUDE_PCI_CONFIG = 1 configures the bridge as a host bridge, and since the ML410 does the configuration in this setup, the ML555 C_INCLUDE_PCI_CONFIG is set to 0. When C_INCLUDE_BAR_OFFSET = 0, the C_IPIFBAR2PCIBAR_* generic(s) are used in address translation instead of IPIFBAR2PCIBAR_* registers. Setting C_IPIFBAR_NUM = 2 specifies that there are two address ranges for PLB to PCI transactions. Setting C_PCIBAR_NUM = 2 specifies that two address ranges are used for PCI to PLB transactions.

Figure 6 provides a functional diagram of the PLBv46 PCI core in the ML555 Development Platform. The functions in the PLBv46 PCI are the PLBv46 Master, PLBv46 Slave, v4.0 PCI Core, and the IPIF/v4.0 Bridge.

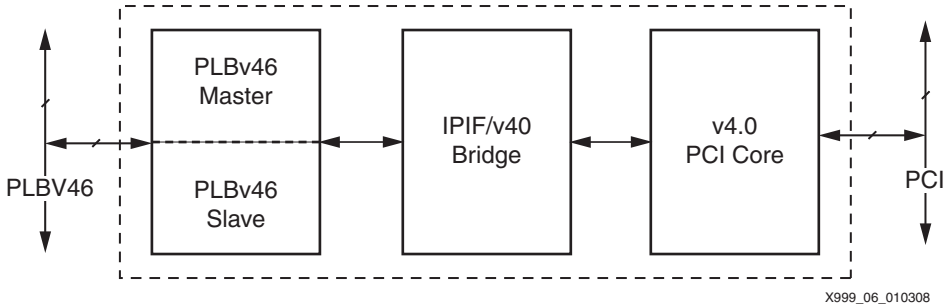


Figure 6: **PLBv46 PCI Functional Diagram**

Configuration of PLBv46 PCI on the ML555 PCI/PCI Express Evaluation Platform

When the ML555 is inserted into the ML410 PCI slot P3, the PLBv46 PCI Bridge in the XC4VFX60 FPGA interfaces to the PLBv46 PCI Bridge in the XC5VLX50T FPGA on the ML555.

To configure the XC4VFX60, connect the Xilinx Download (USB or Parallel IV) cable to the ML410 JTAG port, and use Impact to download the `ML410 ml410_ppc_plbv46_pci/implementation/download.bit` file.

To configure the XC5VLX50T, connect the Xilinx Download (USB or Parallel IV) cable to the ML555 JTAG port, and use Impact to download the `ml555_mb_plbv46_pci/ready_for_download/download.bit` file.

After downloading the XC5VLX50T FPGA bit file, the PCI functionality ML555 PLBv46 PCI is configured using configuration write PCI transactions from the ML410 PLBv46 PCI.

Setting Up the ML555 Communication Terminal

The ML555 differs from other Xilinx boards in that the USB port is used to interface to a communication terminal such as Tera Term. This replaces the serial communication cable/null modem/gender changers used by other Xilinx boards to communicate with a communication terminal. The ML555 contains the Silicon Laboratories CP2102 USB to UART Bridge. A USB Type A to Type B Port Interface cable connects between the PC's USB A port and the ML555 USB B port. The Silicon Laboratories' device driver must be installed to use a communication terminal. The CP210x_Drivers.exe driver is available on the ML555 CD shipped when the ML555 is purchased. The XPS UartLite at address `0x84000000` provides the UART function.

The driver uses the next available serial communication port. To learn the communication port number, connect the USB cable between the PC and ML555, right click on My Computer, and select Properties -> Hardware -> Device Manager. The port number (COM3) is displayed as shown in Figure 7.

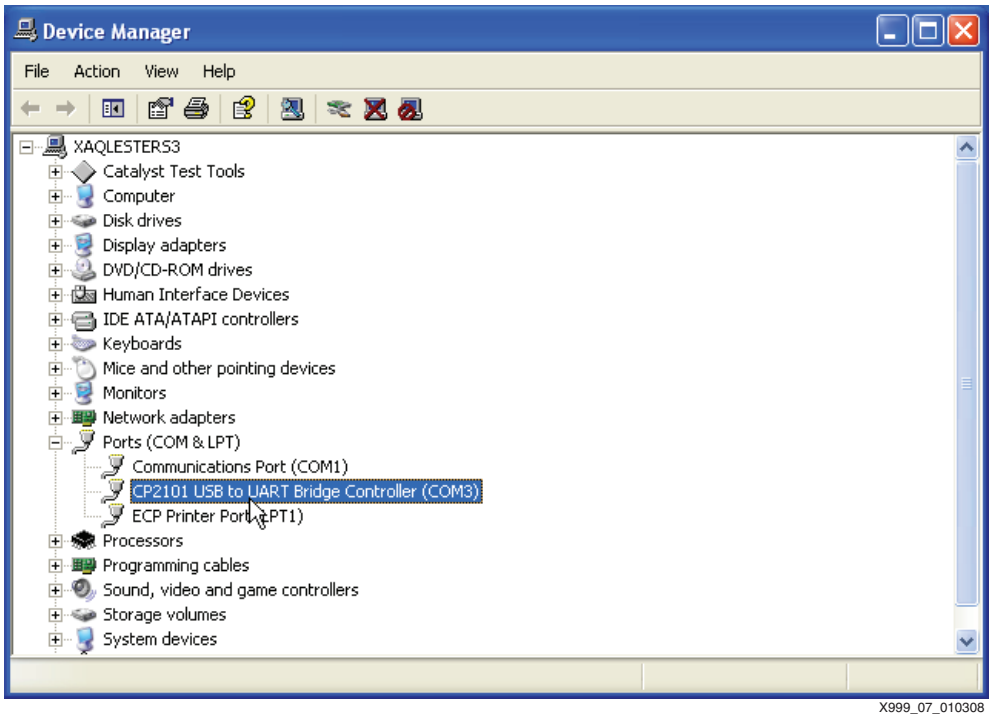


Figure 7: **Device Manager Communication Port**

Figure 8 shows the setup of the TeraTerm utility. Using TeraTerm or a similar serial communications utility, set the TeraTerm Port to **Com3**, Bits per second to **9600**, Data Bits to **8**, Parity to **None,** and Flow Control to **None**. This is necessary to see the results from the software application.

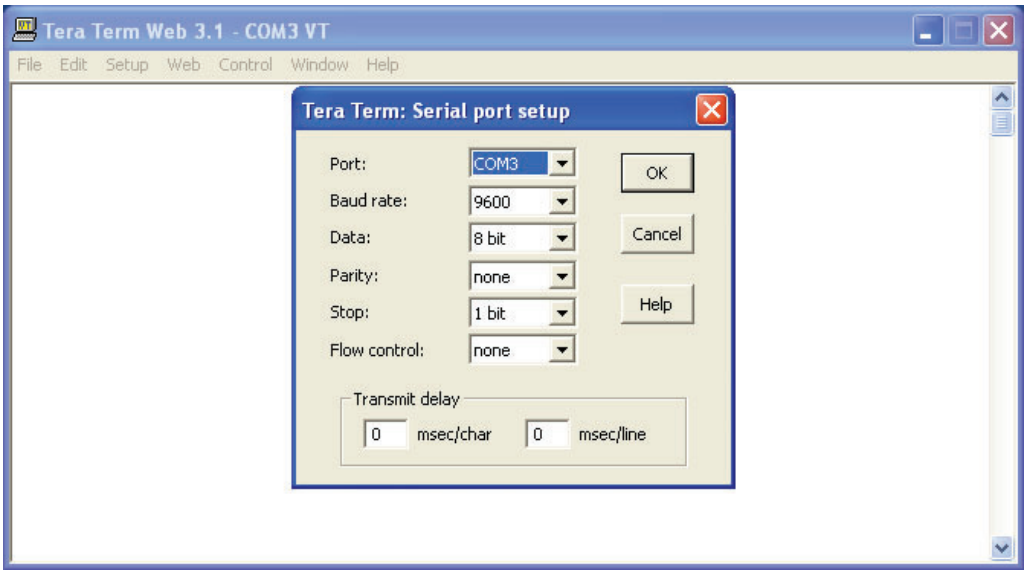


Figure 8: **Tera Term Setup**

Executing the Reference System using the Pre-Built Bitstream and the Compiled Software Applications

Use the following steps to execute the system using files in the `ml555_mb_plbv46_pci/ready_for_download` directory.

1. Change to the `ml555_mb_plbv46_pci/ready_for_download` directory.
2. Use iMPACT to download the bitstream.
   `impact -batch xapp999.cmd`
3. Invoke XMD and connect to and reset the MicroBlaze processor.
   `xmd -opt xapp999.opt`
4. While the PLBv46 PCI in the ML555 can act as the host bridge, this reference design uses the PLBv46 PCI in the ML410 as the host bridge. Connect the JTAG cable to the ML410. Download the bitstream. Download the hello_pci executable file.
   `dow Ml410_ppc_plbv46_pci/ready_for_download/hello_pci.elf`

Executing the Reference System from EDK

Set the environmental variable XIL_PLACE_ALLOW_LOCAL_BUFG_ROUTING=1.

Use the steps below to execute the system using XPS.

1. **File** → **Open** → **system.xmp**.
2. Select **Hardware** → **Generate Bitstream** to generate a bitstream
3. Download the bitstream to the board using **Device Configuration** → **Download Bitstream**.
4. Invoke XMD with **Debug** → **Launch XMD**.

5. While the PLBv46 PCI in the ML555 can act as the host bridge, this reference design uses the PLBv46 PCI in the ML410 as the host bridge. Connect the JTAG cable to the ML410. Download the bitstream. Download the executable by the following command.
   `dow ml410_ppc_plbv46_pci/ready_for_download/hello_pci.elf`

Verifying the Reference Design with the Xilinx Microprocessor Debugger

After downloading the bitstream file and writing to the configuration header, verify that the ML555 reference design is set up correctly.

1. Configure the v4.0 Command Register, Latency Timer, and BAR(s).
2. Read the configuration header.
3. Configure the Command Register, Latency Timer, and BAR(s) of the other devices in the system.
4. Read the configuration headers of the other devices in the system.
5. Perform a memory read of one of the IPIF BARs.
6. Perform a memory write of one of the IPIF BARs.

Verification is done using either Xilinx Microprocessor Debugger (XMD) or the software projects discussed later. TCL scripts using the XMD commands are provided in the `ml555_mb_plbv46_pci/xmd_commands` directory. With the JTAG connection attached to the ML410, the `410_555.tcl` script configures and verifies the ML410 and ML555 PLBv46 PCI cores. To run this script, enter

`xmd -tcl wr_410_555.tcl`

at the command prompt.The XMD commands in the `410_555.tcl` file, partially listed in Figure 9, write to the Configuration Address Port and to the Configuration Data Port to program the Configuration Space Headers. The Command/Status Register, Latency Timer, and Base Address Registers of the ML410 and ML555 PLBv46 PCIs are written and read.

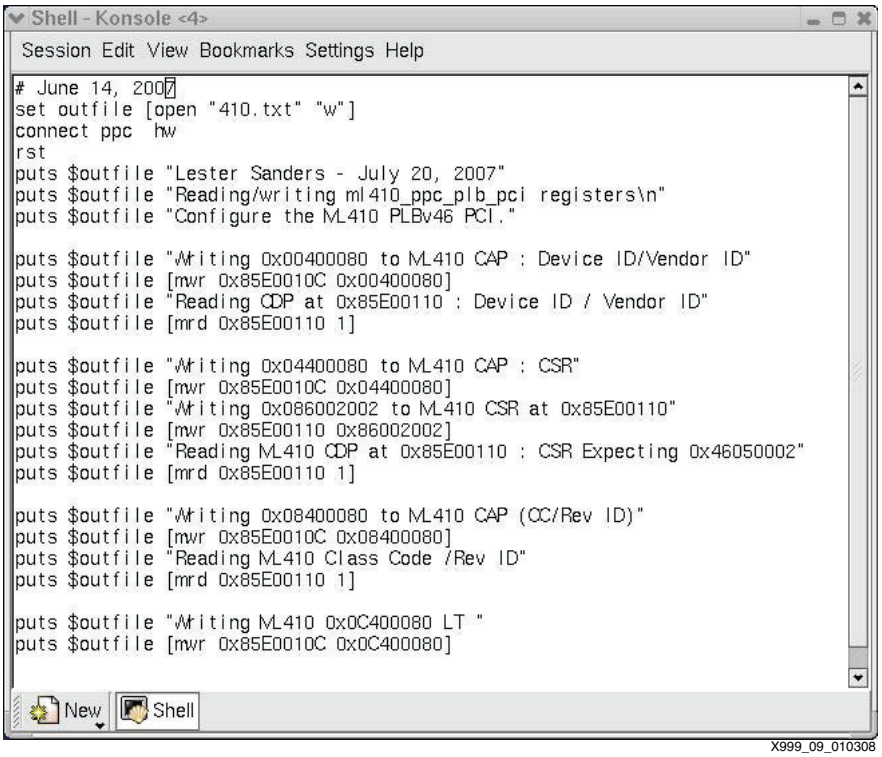

Shell - Konsole <4>
Session Edit View Bookmarks Settings Help
June 14, 2007
set outfile [open "410.txt" "w"]
connect ppc hw
rst
puts $outfile "Lester Sanders - July 20, 2007"
puts $outfile "Reading/writing ml410_ppc_plb_pci registers\n"
puts $outfile "Configure the ML410 PLBv46 PCI."
puts $outfile "Writing 0x00400080 to ML410 CAP : Device ID/Vendor ID"
puts $outfile [mwr 0x85E0010C 0x00400080]
puts $outfile "Reading CDP at 0x85E00110 : Device ID / Vendor ID"
puts $outfile [mrd 0x85E00110 1]
puts $outfile "Writing 0x04400080 to ML410 CAP : CSR"
puts $outfile [mwr 0x85E0010C 0x04400080]
puts $outfile "Writing 0x086002002 to ML410 CSR at 0x85E00110"
puts $outfile [mwr 0x85E00110 0x86002002]
puts $outfile "Reading ML410 CDP at 0x85E00110 : CSR Expecting 0x46050002"
puts $outfile [mrd 0x85E00110 1]
puts $outfile "Writing 0x08400080 to ML410 CAP (CC/Rev ID)"
puts $outfile [mwr 0x85E0010C 0x08400080]
puts $outfile "Reading ML410 Class Code /Rev ID"
puts $outfile [mrd 0x85E00110 1]
puts $outfile "Writing ML410 0x0C400080 LT "
puts $outfile [mwr 0x85E0010C 0x0C400080]
New
Shell
X999_09_010308

Figure 9: **XMD Commands for Configuring PLBv46 PCIs**

Software Projects

The reference system contains the following software projects.

hello_pci. This project enables master transactions, sets the latency timer, defines the bus number/subordinate bus number, and scans the ML410 PCI Bus configuration space headers.

pci_dma. This project runs Direct Memory Access (DMA) operations. The user sets the source address, destination address, and DMA length. This code is used for DMA operations between a variety of source and destination addresses. Figure 10 shows the parameters in pci_dma.c which can be edited to test PCI transactions between different memory regions. The pci_dma.c is run on the ppc405 in the ML410. It can also run on the MicroBlaze processor in the ML555.

```
define MEM_0_BASEADDR 0x20000000
define MEM_1_BASEADDR 0x20002000
..

DMALength = 1024
```

X999_10_010308

Figure 10: **Defining Source and Destination Addresses, Length in pci_dma.c**

DMA Transactions

Some of the XMD scripts and C code examples generate DMA operations. DMA transactions are done by writing to the Control, Source Address, Destination Address, and Length registers of the DMA controller. Table 5 provides the register locations of the XPS Central DMA.

Table 5: **XPS Central DMA Registers**

DMA Register	Address
Control Register	C_BASEADDR + 0x04
Source Address Register	C_BASEADDR + 0x08
Destination Address Register	C_BASEADDR + 0x0C
Length Register	C_BASEADDR + 0x10

The pci_dma.c code consists of the four functions in the functional diagram in Figure 11. The Barberpole Region function provides a rotating data pattern on the memory located at the source address. The Zero Region function sets the memory located at the destination address to all zeroes. The DMA Region function performs a DMA transaction of data located at the source address to the memory at the destination address. The Verify function verifies that data at the source address and destination address are equal.

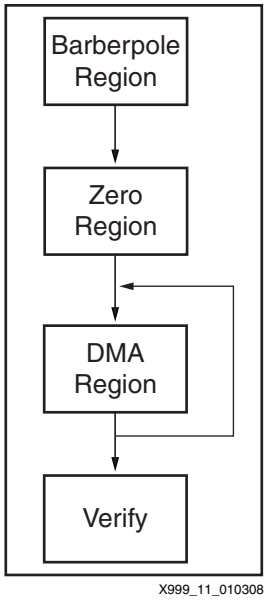


Figure 11: **Functional Diagram of pci_dma.c**

Figure 12 shows the Hyperterminal output when running the `pci_dma/executable.elf`. In the first run, Length is 100. In the second run, Length is 400.

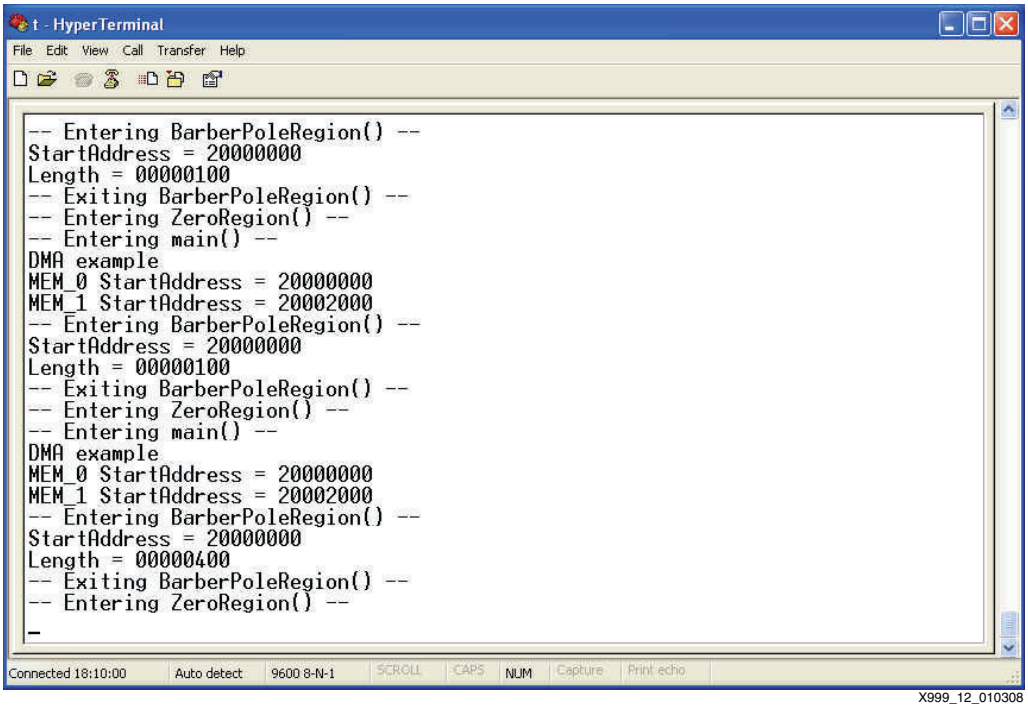
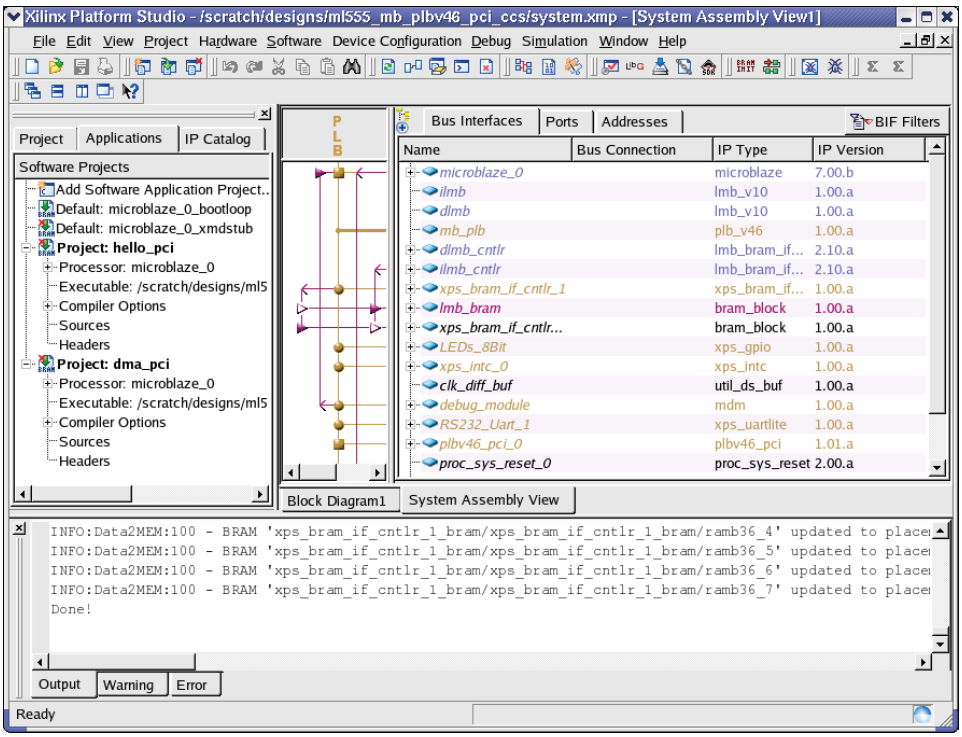


Figure 12: **pci_dma.c output**

Running the Applications

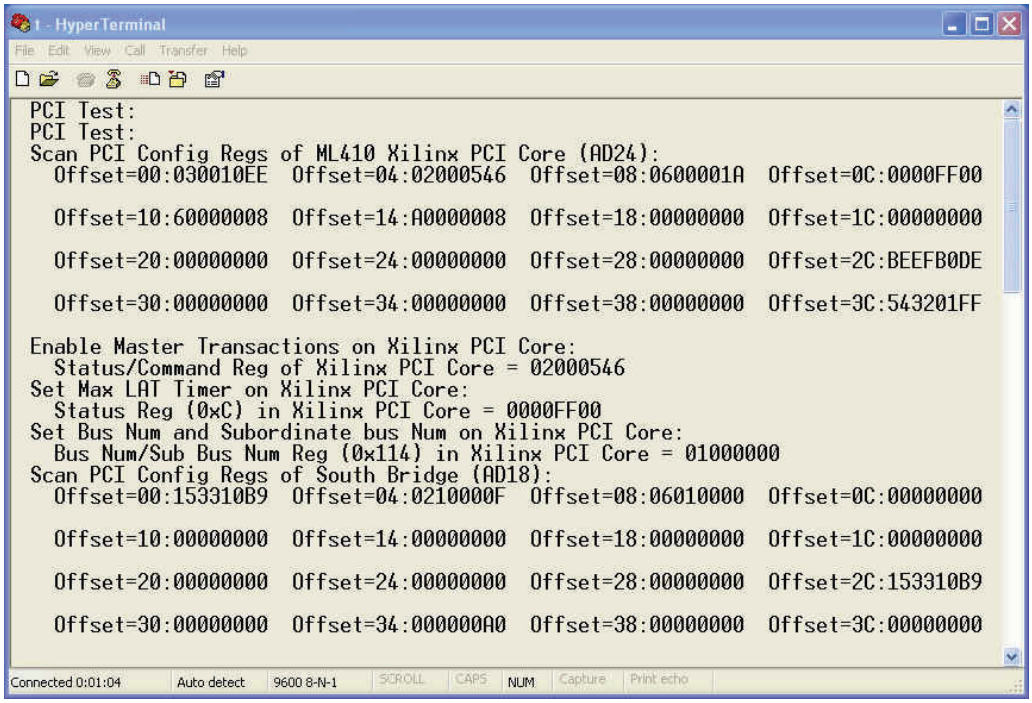
The selection of the hello_pci is shown in Figure 13. Make the hello_pci project active and the remaining software projects inactive.



X999_13_010308

Figure 13: **Selecting the hello_pci Software Project**

From XPS, start **XMD** and enter `connect ppc hw` and `rst` at the XMD prompt. Invoke GDB and select Run to start the application as shown in Figure 14.


t - HyperTerminal
File Edit View Call Transfer Help
PCI Test:
PCI Test:
Scan PCI Config Regs of ML410 Xilinx PCI Core (AD24):
Offset=00:030010EE Offset=04:02000546 Offset=08:0600001A Offset=0C:0000FF00
Offset=10:60000008 Offset=14:A0000008 Offset=18:00000000 Offset=1C:00000000
Offset=20:00000000 Offset=24:00000000 Offset=28:00000000 Offset=2C:BEEFB0DE
Offset=30:00000000 Offset=34:00000000 Offset=38:00000000 Offset=3C:543201FF
Enable Master Transactions on Xilinx PCI Core:
Status/Command Reg of Xilinx PCI Core = 02000546
Set Max LAT Timer on Xilinx PCI Core:
Status Reg (0xC) in Xilinx PCI Core = 0000FF00
Set Bus Num and Subordinate bus Num on Xilinx PCI Core:
Bus Num/Sub Bus Num Reg (0x114) in Xilinx PCI Core = 01000000
Scan PCI Config Regs of South Bridge (AD18):
Offset=00:153310B9 Offset=04:0210000F Offset=08:06010000 Offset=0C:00000000
Offset=10:00000000 Offset=14:00000000 Offset=18:00000000 Offset=1C:00000000
Offset=20:00000000 Offset=24:00000000 Offset=28:00000000 Offset=2C:153310B9
Offset=30:00000000 Offset=34:000000A0 Offset=38:00000000 Offset=3C:00000000
Connected 0:01:04
Auto detect
9600 8-N-1
SCROLL
CAPS
NUM
Capture
Print echo

X999_14_010308

Figure 14: **Running hello_pci in gdb**

Using Chipscope Analyzer with PLBv46 PCI

The Chipscope Analyzer is used to debug hardware problems. Debugging can be done at either the system or PLBv46 PCI core level. To analyze PLBv46 PCI internal signals, insert the Chipscope analyzer cores into pci32_bridge_wrapper.ngc. To analyze signals involving multiple cores, insert the Chipscope analyzer cores into system.ngc. The flow for using the two debugging methods differs. Below, an outline of the steps for debugging at the system level is provided. This is followed by a detailed list of the steps used for debugging at the core level.

Inserting Chipscope Analyzer at the System Level

The following steps insert the Chipscope analyzer cores into the system.

1. In XPS, select **Hardware** → **Generate Netlist.**
2. From the command prompt in the ml555_mb_plbv46_pci directory, run

```
ngcbuild -i system.ngc system2.ngc
```

3. Invoke Chipscope Inserter. To specify the input file in Inserter's **Input Design Netlist** dialog box, browse to the system2.ngc file created in step 2. The `ml555_mb_plbv46_pci_scs.cdc` defines the clock, trigger, and data signals. Add signals if desired. Generate the ICON and ILA cores.
4. From ml555_mb_plbv46_pci/implementation, copy the Inserter Output Design Netlist file, usually system2.ngo, to system.ngc.
5. In XPS, run **Hardware** → **Generate Bitstream**.

The `chipscope/ml555_mb_plbv46_pci_scs.cpj` is a saved project for this debug session.

Inserting Chipscope at the PLBv46 PCI Core Level

The `ml555_mb_plbv46_pci/chipscope/plbv46_pci.cdc` file is used to insert a Chipscope analyzer ILA core into the pci32_bridge_wrapper core. Run the following steps to insert a core and analyze PLBv46 PCI problems with Chipscope analyzer.

1. Invoke XPS. Select **Hardware** → **Generate Netlist**.
2. Run **Start** → **Programs** → **ChipScope Pro** → **ChipScope Inserter**
3. From Chipscope Inserter, run **File** → **Open Project chipscope/plbv46_pci_ccs.cdc**.

Figure 15 shows the Chipscope analyzer Inserter setup GUI after **File Open** → **plbv46_pci_ccs.cdc**.

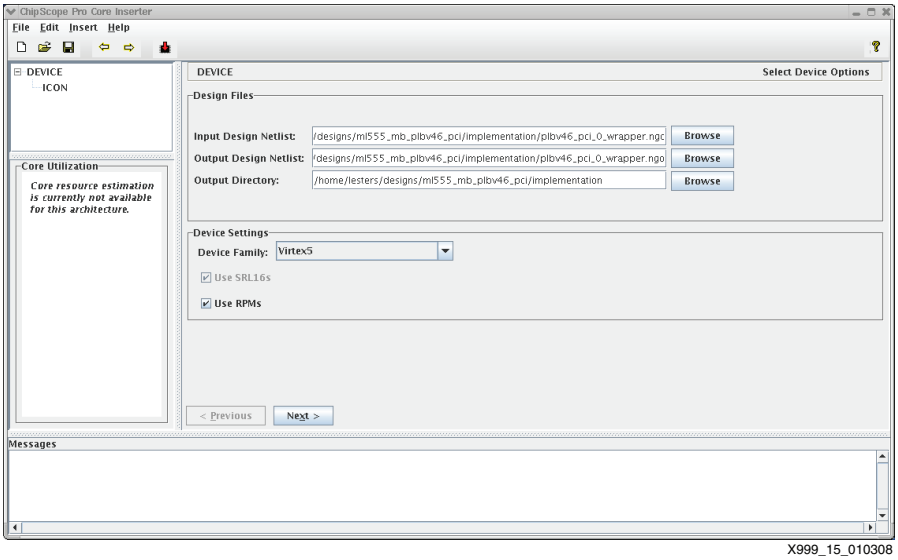


Figure 15: **Chipscope Analyzer Inserter Setup**

The PCI_Monitor signals are the PCI bus signals: AD, CBE, and the remaining PCI Bus signals. Table 6 defines the functionality of the PCI_Monitor signals. The Filter Pattern *PCI_Monitor* is used to locate the PCI signals.

Table 6: **PCI Monitor Signals**

Bit Position	PCI Signal
0	FRAME_N
1	DEVSEL_N
2	TRDY_N
3	IRDY_N
4	STOP_N
5	IDSEL_int
6	INTA
7	PERR_N
8	SERR_N
9	Req_N_toArb
10	PAR
11	REQ_N
12:43	AD
44:47	CBE

4. The GUI for making net connections is shown in Figure 16. Click **Next** four times to move to the Modify Connections window. The signals defined by `plbv46_pci_ccs.cdc` are displayed. In some analyses, it is necessary to add signals. To add signals, click on **Modify Connections**. The Filter Pattern can be used to find net(s). As an example of using the

Filter Pattern, enter `*ack*` in the dialog box to locate acknowledge signals such as SI_AddrAck. In the Net Selections area, select either **Clock**, **Trigger**, or **Data signals**. Select the **net** and click **Make Connections**.

If **Clock**, **Trigger**, and/or **Data** signals are displayed in red, the signal is unconnected. To correct unconnected signals, highlight the signal, select a net from the left `Net Name` window, and click on **Make Connections**.

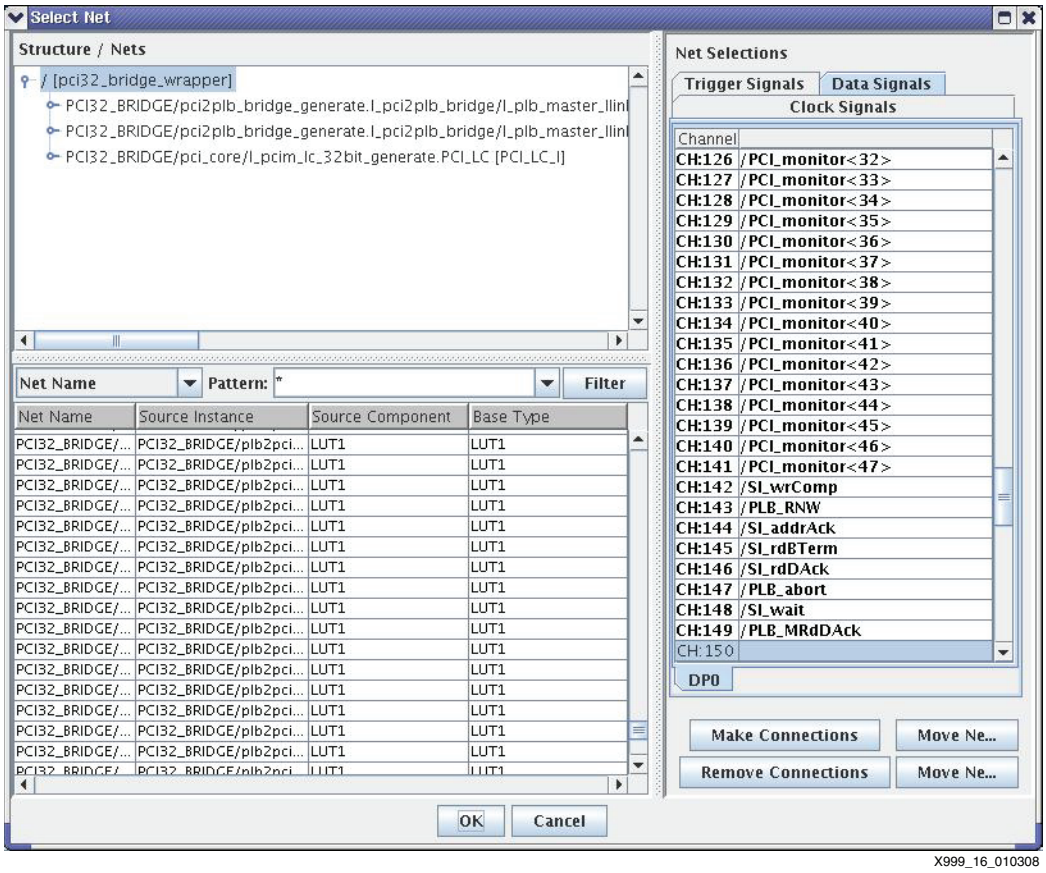


Figure 16: **Making Net Connections in Chipscope Analyzer Inserter**

5. Click **Insert** to insert the core into pci32_bridge_wrapper.ngo.

In the `ml555_mb_plbv46_pci/implementation` directory, copy `pci32_bridge_wrapper.ngo` to `pci32_bridge_wrapper.ngc`.

6. Set the environmental variable XIL_PLACE_ALLOW_LOCAL_BUFG_ROUTING=1.
7. In XPS, run **Hardware** → **Generate Bitstream** and **Device Configuration** → **Download Bitstream**. Verify that the file size of the pci32_bridge_wrapper.ngc with the inserted core is significantly larger than the original version.

Note: Do not rerun **Hardware** → **Generate Netlist**, as this overwrites the `implementation/pci32_bridge_wrapper.ngc` produced by the step above.

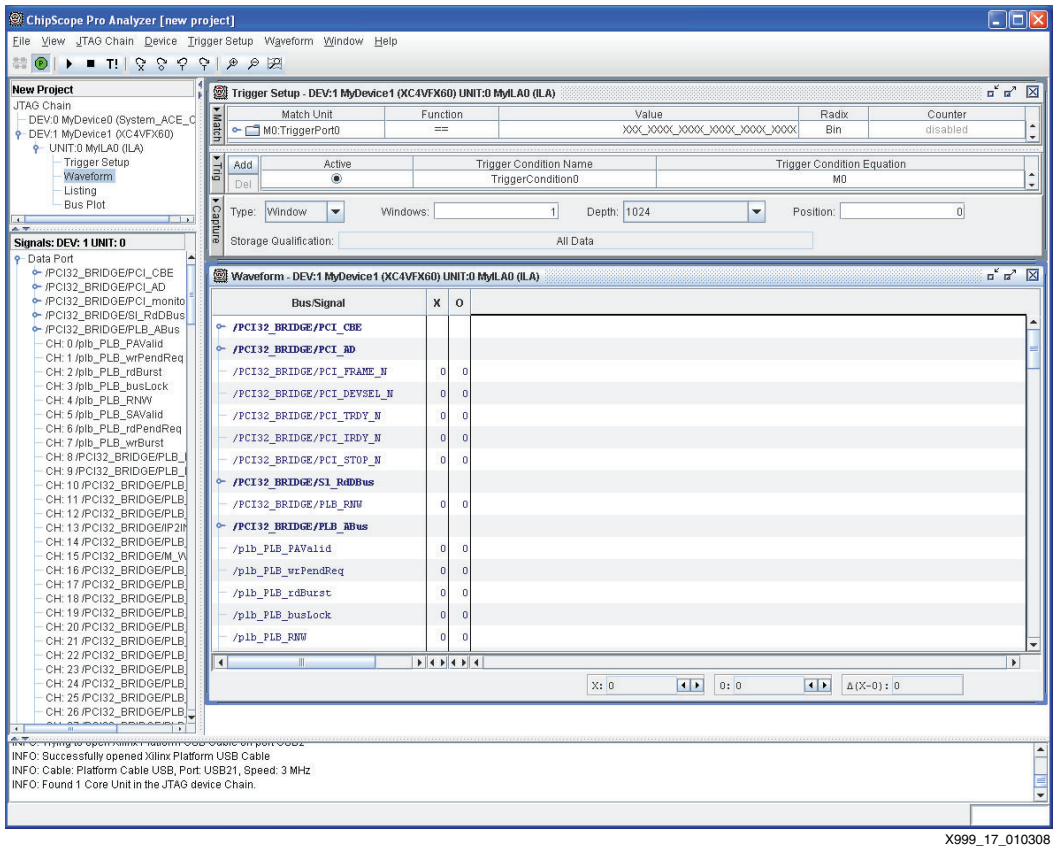
8. Invoke ChipScope Pro Analyzer by selecting from Windows start menu

Start → **Programs** → **ChipScope Pro** → **ChipScope Pro Analyzer**

Click on the Chain icon located at the top left of Analyzer GUI. Verify that the message in the transcript window indicates that an ICON is found.

9. The ChipScope Pro Analyzer waveform viewer displays signals named `DATA*`. To replace the `DATA*` signal names with the familiar signal names specified in Chipscope analyzer Inserter, select **File** → **Import** and browse to plbv46_pci_ccs.cdc in the dialog box.

The Analyzer waveform viewer is more readable when buses rather than discrete signals are displayed. Select the 32 **PLB_ABus<*>** signals, click the right mouse button, and select **Add to Bus** → **New Bus**. With **PLB_ABus<0:31>** in the waveform viewer, select and delete the **32 discrete PLB_ABus<*>** signals. Repeat this for the PLBv46 data buses. Make PCI Bus signals by creating a new bus for PCI_Monitor(44:47). Rename PCI_Monitor(44:47) PCI_CBE. Create a bus for PCI_Monitor(12:43) and rename it PCI_AD. The signals are displayed as buses in Figure 17.



X999_17_010308

Figure 17: **Creating Buses in ChipScope Analyzer**

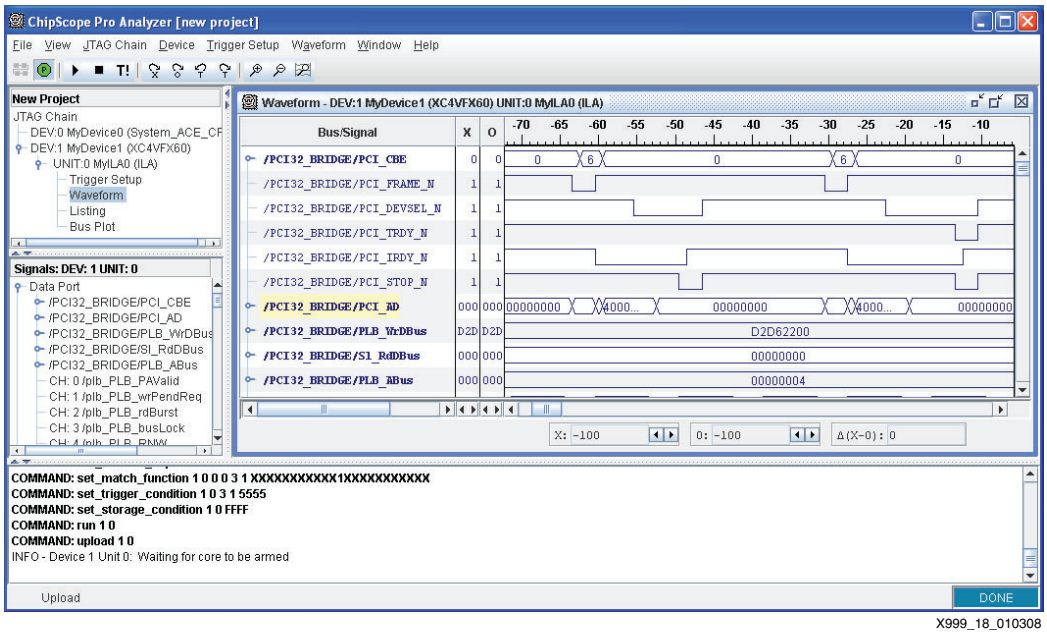
Note: The Reverse Bus Order operation option is useful for analyzing bus signals in Analyzer.

10. Set the trigger in the Trigger Setup window. The trigger used depends on the problem being debugged. For example, if debugging a configuration transaction from the ML410 PLBv46, trigger on an PLB address of C_BASEADDR + 0x10C. If debugging a problem configuring from the PCI side, trigger on the PCI_Monitor(44:47) for a configuration write on PCI_CBE. Simpler triggers are PCI_FRAME_N (PCI_Monitor_0) for PCI transactions and PA_Valid for PLBv46 transactions. Arm the trigger by selecting **Trigger Setup** → **Arm**, or clicking on the **Arm** icon.
11. Run **XMD** or **GDB** to activate trigger patterns which cause Chipscope Pro Analyzer to display waveform output. For example, set the trigger to PA_Valid = 1, and run

```
xmd -tcl xmd_commands/410_555.tcl
```

at the command prompt. This produces signal activity in Analyzer's waveform viewer.

12. The Chipscope Pro Analyzer results are analyzed in the waveform window, as shown in Figure 18. This figure shows the bus signals generated in Step 10. To share the results with remote colleagues, save the results in the waveform window as a Value Change Dump (vcd) file. The vcd files can be translated and viewed in most simulators. The `vcd2wlf` translator in ModelSim reads a vcd file and generates a ModelSim waveform log file (wlf) file for viewing in the ModelSim waveform viewer. The vcd file can be opened in the Cadence Design System, Inc. Simvision design tool by selecting **File** → **Open Database**.



X999_18_010308

Figure 18: **ChipScope Analyzer Results**

After running the Chipscope Pro Analyzer, it is sometimes necessary to revise the Trigger or Data nets used in a debug operation in Chipscope Pro Inserter. This procedure is made easier if Inserter and Analyzer projects are saved. The projects can be re-opened, allowing only the edits to be made.

The plbv46_pci_ccs.cpj is a project used for this debug session.

Reference Design Matrix

The reference design matrix is shown in Table 7.

Table 7: **Reference Design Matrix**

General	
Developer Name	Xilinx
Target devices (stepping level, ES, production, speed grades)	Virtex-5 LX50T
Source code provided	No
Source code format	VHDL
Design uses code/IP from an existing reference design/application note, 3rd party, or CORE Generator software	No
Simulation	
Functional simulation performed	No

Table 7: **Reference Design Matrix**

Timing simulation performed	No
Testbench used for functional simulations provided	No
Testbench format	N/A
Simulator software used/version (i.e., ISE software, Mentor, Cadence, other)	N/A
SPICE/IBIS simulations	No
Implementation	
Synthesis software	XST
Implementation software tools used/versions	ISE9.2i SP4
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	ML410/ML555

References

1. UG159 *LogiCORE IP Initiator/Target v3.1 for PCI User Guide*
2. UG262 *LogiCORE IP Initiator/Target v4.5 for PCI User Guide*
3. UG044 *ChipScope ILA Tools Tutorial*
4. UG241 *OPB PCI User Manual*
5. UG201 *Virtex-5 ML555 PCI/PCI Express Development Kit*
6. UG085 *ML410 Embedded Development Platform User Guide (v1.2)*
7. XAPP1001 *Reference System: PLBv46 PCI Using the ML410 Embedded Development Platform*
8. XAPP998 *PCI Bus Performance Measurements using the Vmetro Bus Analyzer*

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
2/8/08	1.0	Initial Xilinx release.

Notice of Disclaimer

Xilinx is disclosing this Application Note to you “AS-IS” with no warranty of any kind. This Application Note is one possible implementation of this feature, application, or standard, and is subject to change without further notice from Xilinx. You are responsible for obtaining any rights you may require in connection with your use or implementation of this Application Note. XILINX MAKES NO REPRESENTATIONS OR WARRANTIES, WHETHER EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL XILINX BE LIABLE FOR ANY LOSS OF DATA, LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES ARISING FROM YOUR USE OF THIS APPLICATION NOTE.